2차원 패턴의 볼록 헐 알고리즘

홍기천[†] · 오일석[††]

요 약

본 논문에서는 2 차원 패턴을 위한 볼록 헐(convex hull) 알고리즘을 제안한다. 알고리즘은 크게 후보 볼록점 추출과 최종 볼록점 추출의 두 단계로 나뉜다. 첫 번째 단계에서는 볼록 헐의 볼록점이 될 수 없는 점들을 최대한 간단한 연산을 사용하여 제거함으로써 속도의 향상을 기한다. 두 번째 단계에서는 첫 번째 단계에서 구해진 후보 볼록점을 대상으로 최종 볼록 헐을 구한다. 이 방법은 매우 간단한 연산으로 구성되어 있기 때문에 수행 속도면에서 향상을 가져왔다. 실험 결과, 본 논문의 방법이 기존에 사용되던 두 개의 볼록 헐 알고리즘보다 2배내지 3배의 빠른 수행 속도를 보였다.

A Convex Hull Algorithm for 2D Patterns

Ki-Cheon Hong[†] and Il-Seok Oh[††]

ABSTRACT

This paper proposes a convex hull algorithm for 2D patterns. The proposed algorithm is divided into 2 steps; candidate convex point extraction and final convex point extraction. First step removes as many points as possible that cannot be convex points using simple operation. Second step computes final convex hull of 2D patterns. This method accelerates execution time, since it consists of simple operations. Experimental results show that the proposed method is faster than other 2 methods in speed.

1. 서 론

컴퓨터를 이용하여 어떤 물체(object)의 모양을 분석하는 작업은 영상 검색[1], 사무 자동화, 공장 자동화[2], 의료 분야[3], 생체공학, 패턴 인식과 같은 분야에 널리 사용되고 있다. 실세계에 존재하는 모든 물체는 고유의 모양 특성을 가지고 있다. 이러한 모양 특성을 최대한 이용하면 물체 인식에 많은 도움이 될 것이다. 예를 들어, 공장에서 생산되는 물품에서 불량품을 골라내기 위해서 모양을 분석할 수도 있고 문서내에 있는 문자들을 분류하기 위해 모양을 분석할 수 있다. 이러한 작업을 컴퓨터가 자동으로 하기 위해서는 분별력이 좋은 물체의 모양 특징을 추출해야 한다. 이러한 모양 특징을 추출하기 위한 방법으로 여러가지 특징 추출 방법[4,5]이 있을 수 있지만 그 중 유용하게 사용할 수 있는 것이 볼록 헐 알고리즘이다. 이러한 볼록 헐 알고리즘이 가져야할 조건으로는 실제 산업 현장에서 사용될 수 있을 정도의 정확성과 빠른 수행 능력이다.

볼록 헐 알고리즘은 물체를 둘러싸고 있는 최소의 볼록 다각형을 구하는 것이다. 이제까지 볼록 헐을 구하는 논문이 많이 발표되어 있다[6-9]. 이 논문들이 사용한 입력으로는 그림 1과 같이 임의로 분포된 점들의 집합[7]과 2D 영상으로 나눌 수 있다. 2D 영상은 다시 입력 데이터의 자료구조에 따라 2D 배열 사용 방법[6], 단순 다각형의 꼭지점 사용 방법[8], 체인 코드 사용 방법[9]으로 나눌 수 있다.

[8]은 단순 다각형의 꼭지점 좌표를 사용하여 다각형의 볼록 헐을 구하였다. 이 연구는 다각형의 꼭지점들을 시계 반대 방향으로 정렬한 후, 이 점들의 Y축과 X축 좌표값을 수식에 적용하여 볼록 헐을 구하였다. 이 수식의 결과에 따라서 점들의 볼록성 여부

[†] 정회원, 전주교육대학교 컴퓨터교육과 전임강사
[††] 전북대학교 컴퓨터과학과 교수

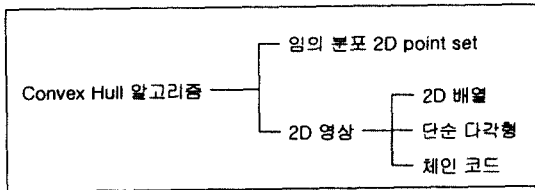



그림 1. 볼록 헐 알고리즘 분류

를 판단하였다.

[9]는 체인 코드를 이용하여 단순 다각형의 볼록 헐을 구하였다. 이 연구는 어떤 물체(blob)의 볼록 헐을 구하기 위하여 외곽선의 체인 코드값(0~7)을 사용하였다. 이 알고리즘은 크게 두 단계로 나뉜다. 첫 번째 단계는 n+1 번째 점의 체인코드 값==((n 번째 점의 체인코드 값+(5 or 6 or 7)) mod 8)의 규칙을 정의하여, 이를 만족하면 n+1번째 점은 볼록 헐을 구성하는 후보 점으로 결정한다. 즉, 첫 번째 단계는 체인 코드 중 국소적으로 볼록한 점들로 구성되는 후보점들을 추출하는 것이 목적이다. 두 번째 단계는 어떤 물체(blob)가 가질 수 있는 6개의 경로 형태를 정의하여 첫 번째에서 결정된 후보점들이 6개의 형태 중 하나에 속하면 볼록 헐을 구성하는 점이 아니라고 판단하여 제거한다. 두 번째 단계를 더 이상 제거할 점들이 없을 때까지 모든 후보점들을 대상으로 반복하였다. 두 번째 단계는 국소적으로 볼록한 후보점들 중 전역적으로 볼록한 점들만 추출하는 단계라고 할 수 있다. 이 방법은 볼록 헐을 구하기 위해서 정수 뺄셈과 덧셈만을 이용하여 속도를 향상시키려 노력하였다.

그러나 위의 두 논문을 필기 한글 패턴에 대해서 구현해 본 결과, 속도가 결코 빠른 편이 아니라는 것을 알았다. 필기 한글을 적용한 이유는 필기 한글에는 다양한 모양의 패턴이 포함되어 있으므로 볼록 헐 알고리즘의 동작과 성능을 평가하는데 적합하다고 할 수 있기 때문이다. [8]은 하나의 수식으로 모든 점들의 볼록성을 계산하였다. 이것은 많은 양의 수식 연산이 필요하게 된다. [9] 첫 번째 단계에서 볼록 헐을 구성하지 않는 점들을 충분히 제거해야 되는데 그렇지 못하였다. 이러한 문제들 때문에 두 개의 알고리즘의 속도의 효율성이 떨어진 것이다.

위의 두 논문의 방법은 나름대로의 장점을 가지고 있으나 속도를 향상시킬 여지가 남아있다. 이에 착안하여 본 논문에서는 위의 두 가지 방법이 가지고 있는 장점을 각각 채택하여 속도가 빠른 볼록 헐 알고리즘을 개발하였다. 본 논문에서는 볼록 헐을 구성하지 않는 점들을 최대한 먼저 제거한다는 생각에서 [9]의 논문에서와 같이 두 단계로 나누었다. 첫 번째 단계에서는 입력 데이터의 외곽선 점들 중, 볼록 헐을 구성하지 않는 점들을 최대한 제거하여 후보점을 추출하였다. 두 번째 단계에서는 [8]의 논문에서 제안한 수식을 본 논문에 적용하였다. 이렇게 함으로써 두 번째 단계에서는 최소한의 점들을 가지고 수식 연산을 하도록 하였기 때문에 전체적인 알고리즘의 속도를 향상시킬 수 있었다.

제안한 알고리즘은 크게 후보 볼록점 추출과 최종 볼록점 추출의 두 단계로 나뉜다. 첫 번째 단계는 패턴의 외곽선 위에 패턴을 포함하는 bounding box를 씌워서 bounding box의 경계선과 겹치는 점들을 추출한다. 추출된 점들 중, 서로 인접해 있는 점들을 하나의 그룹으로 묶는다. 이때, 그룹과 그룹사이에 존재하는 점들의 집합을 하나의 세그먼트라고 한다면, 이 세그먼트의 Y축과 X축의 좌표값을 정의된 수식에 넣는다. 이 수식의 결과 값에 따라 볼록하지 않는 점들은 모두 제거한다.

두 번째 단계는 제거되지 않고 남아있는 점들의 Y축과 X축의 좌표값을 정의된 수식에 집어넣어 최종적인 입력 패턴의 볼록 헐을 구한다. 이 알고리즘은 패턴의 외곽선 점들의 체인 코드의 코드값(0~7)보다는 시계 반대 방향으로 외곽선 점들의 Y축과 X축 좌표값만을 이용하여 볼록 헐을 구하였다. 좌표값을 사용한 이유는 체인 코드값을 사용하면 어느 한 점의 8 방향에 있는 점들만 처리할 수 있기 때문이다. 즉, 8 방향에 들어있지 않는 점들에 대해서는 연산을 할 수 없다.

제 2 장에서는 제안한 알고리즘 자세히 기술한다. 제 3 장에서는 제안한 알고리즘의 실험 결과를 보인다. 또한 다른 볼록 헐 알고리즘과의 속도를 측정하여 비교하였다. 제 4 장에서는 결론 및 향후 연구 과제에 대해서 기술한다.

2. 볼록 헐 알고리즘

이 장에서는 제안한 볼록 헐 알고리즘을 자세히 기술한다. 이 알고리즘은 크게 후보 볼록점 추출 단계와 최종 볼록점 추출과 같이 두 단계로 나누어진다. 알고리즘을 두 단계로 설정한 이유는 첫 번째 단

계에서 볼록 헐에 참여하지 않는 점들을 미리 최대한 제거함으로써 두 번째 단계에서 최소한의 연산을 수행하기 위함이다. 그럼으로써 전체적인 알고리즘의 속도를 향상시킬 수 있다.

2.1 후보 볼록점 추출

첫 번째 단계를 개념적으로 기술하면, 두 점을 잇는 가상적인 선분의 바깥쪽에 위치한 점들만 남기고 선분의 안쪽에 위치한 점들을 제거하는 과정이라고 말할 수 있다. 여기에서 선분의 바깥쪽이란 선분을 기준으로 입력 패턴의 배경(background) 방향을 의미하고, 안쪽이란 입력 패턴의 전경(foreground) 방향을 의미한다. 그러면 입력 패턴의 볼록 헐을 구성하는 후보점들만 남게된다. 이 후보점들은 두 번째 단계의 입력으로 사용된다. 첫 번째 단계는 외곽선상의 점들에 대해서 단지 한번만 수행한다.

맨 처음 입력 패턴(그림 2(a))의 외곽선(그림 2(b))을 구한 후, 이 외곽선위에 패턴을 포함하는 가장 작은 직사각형의 bounding box를 덮어씌운다. 그러면 외곽선과 bounding box가 겹치는 점들을 추출할 수 있다. 이 때 이 점들을 그룹으로 나눌 수 있다. 이것을 $G=\{G_i \mid 1 \le i \le n\}$라 하자. 그림 2(c)는 그림 2(b)에 bounding box를 씌워서 겹치는 점들(시계 반대 방향으로 영어 알파벳 표시)을 나타낸다. 패턴 'ㄹ'은 $G_1=\{a, b, c\}$, $G_2=\{d, e, f, g, h, i, j\}$, $G_3=\{k, l\}$, $G_4=\{m, n\}$와 같이 4 개의 그룹으로 나눌 수 있다. 또한 G_i와 G_{i+1} 사이에 있는 점들의 집합을 세그먼트라 한다면, 그림 2(a)의 패턴 'ㄹ'은 4 개의 세그먼트 S_1, S_2, S_3, S_4로 나눌 수 있다. 그림 2(d)는 각 세그먼트에 속하는 화소를 세그먼트 번호로 표시하였다. 패턴 'ㄹ'을 요약하면 다음과 같다.

P : 전체 패턴

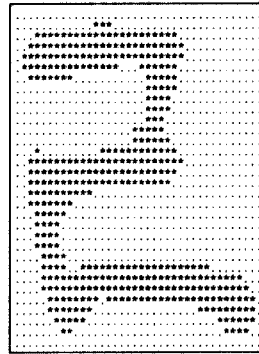
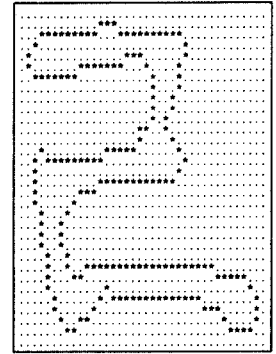
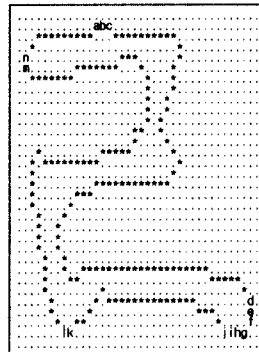
$P = G \cup S$ and $G \cap S = \emptyset$

$G = \{ \{a, b, c\}, \{d, e, f, g, h, i, j\}, \{k, l\}, \{m, n\} \}$

$S = \{S_1, S_2, S_3, S_4\}$

$S_1 = \{(y_i, x_i) \mid s_{c+1} \le i \le s_{d-1}\}$, $S_2 = \{(y_i, x_i) \mid s_{j+1} \le i \le s_{k-1}\}$,

$S_3 = \{(y_i, x_i) \mid s_{l+1} \le i \le s_{m-1}\}$, $S_4 = \{(y_i, x_i) \mid s_{n+1} \le i \le s_{a-1}\}$

이렇게 구해진 세그먼트 S_i의 원소들에 대해서 후보점 여부를 검사한다. 그림 2(d)에서 S_1을 예로

(a) 입력 패턴 (b) 외곽선

(c) bounding box와 겹치는 점 (d) 4 개의 세그먼트

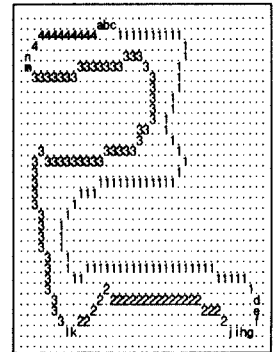



그림 2. 패턴 'ㄹ'의 세그먼트

들면, 집합 $S_1 = \{ (2,15), (2,16), (2,17), \ldots\ldots (26,35)\}$이다. 또한 집합 S_1의 바로 앞의 점 $s_c = (1,14)$이고, S_1의 바로 뒤의 점이 $s_d = (27,36)$이다. 이러한 값들이 주어지면 다음과 같은 연산을 수행한다.

```
(y1, x1) = sc = (1,14);
(y2, x2) = sd = (27,36);
a = y2-y1; b = x1-x2; c = y1*x2-y2*x1;
for( j = 0; j < S1의 원소 개수; j++ ) {
   y = yj; x = xj;
   if ( (a*x+b*y+c) > 0 )
      then { (yj, xj) is a convex point; mark this
      point as NOT_REMOVE; }
   else    { (yj, xj) is not a convex point; mark
   this point as REMOVE; }
}
```

위의 과정을 기술하면 다음과 같다. 두 점 s_c와 s_d는 집합 S_1의 앞과 뒤의 화소이다. 변수 a, b, c는 이

두 점으로부터 계산된다. 집합 S_1의 모든 원소 (y_j, x_j)의 값을 변수 y와 x에 복사한다. 이 값들을 가지고 직선의 방정식 a*x+b*y+c를 계산한다. 만약 이 값이 0보다 크면 (y_j, x_j)는 직선보다 바깥쪽에 위치한 점이다. 즉, 직선을 기준으로 입력 패턴의 배경 방향에 위치한 점이다. 그래서 볼록 헐을 구성하는 후보점이라고 판단하고 "NOT_REMOVE"라는 속성을 부여한다. 그렇지 않고 0보다 작으면 이 점은 직선보다 안쪽에 위치한 점이다. 즉, 직선을 기준으로 입력 패턴의 전경 방향에 위치한 점이다. 그래서 이 점은 후보점이 아니라고 판단하여 "REMOVE"라는 속성을 부여한다. "REMOVE"라는 속성은 두 번째 단계에서 "이 점은 검사 대상에서 제외한다"라는 것을 명시적으로 알려주는 역할을 한다.

그림 3은 집합 S_1, S_2, S_3, S_4에 이 과정을 수행한 후 남은 후보점('$'로 표시)들을 보여준다. 세그먼트 번호가 그대로 남아있는 화소는 외곽선 점들 중, "REMOVE"의 속성을 가지며, 두 번째 단계에서 더 이상의 검사를 하지 않는다. 그림 3에서 보는 바와 같이 첫 번째 단계만으로도 많은 점들이 제거되었음을 알 수 있다. 특히 입력 패턴 'ㄹ'에서 오목한 부분(bay)에 위치한 화소가 모두 제거되었다. 제거되지 않고 남은 후보점들은 "NOT_REMOVE"의 속성을 가지고 있으며, 최종적인 볼록 헐을 구하기 위하여 다음의 두 번째 단계의 입력으로 사용된다.

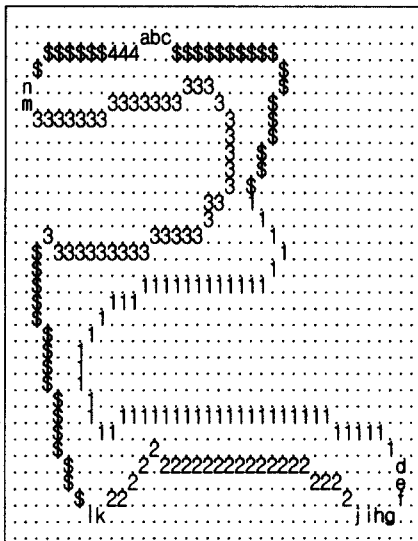



그림 3. 후보점 추출

2.2 최종 볼록점 추출

두 번째 단계를 개념적으로 기술하면, 이 단계는 검사하고자 하는 점이 앞과 뒤 원소를 잇는 가상적인 선분의 바깥쪽에 있는지, 아니면 안쪽에 있는지를 판단하는 것이다. 안쪽과 바깥쪽은 선분을 기준으로 하여 입력 패턴의 배경 방향이면 바깥쪽, 전경 방향이면 안쪽으로 정의한다. 이를 S_1, S_2, S_3, S_4에 적용하면 최종적으로 입력 패턴의 볼록 헐을 구성하는 점들만 남게된다.

이 단계는 첫 번째 단계에서 추출된 후보점들을 대상으로 최종적인 입력 패턴의 볼록 헐을 구하는 단계이다. 후보점들의 개수가 적으면 적을수록 두 번째 단계의 수행 시간은 줄어든다. 그림 3에서 S_1에서 추출된 후보점들을 예로 들어보자. 첫 번째 단계를 수행한 후, 남은 S_1의 후보점의 개수는 18개이다. 이 후보점들에 대해서 다음과 같은 연산을 수행한다.

```
TAG=FALSE; start=sc; end = sd;
while(!TAG) {
   TAG=TRUE;
   for(j=start+1; j<=end-1; j++) {
      select_3_vertices(); // select 3 vertices
      (ya, xa), (yj, xj), (yc, xc)
      if( (xa-xc)*(yc-yj) + (ya-yc)*(xj-xc) < 0)
         then { (yj, xj) is a convex point; mark
         this point as NOT_REMOVE; }
      else { (yj, xj) is not a convex; mark this
      point as REMOVE; TAG=FALSE; }
      }
}
```

위 과정을 기술하면 다음과 같다. 두 점 s_c와 s_d는 집합 S_1의 앞과 뒤의 화소이다. select_3_vertices()함수는 볼록성을 검사할 점(y_j, x_j)을 기준으로 앞 원소(ya, xa)와 뒤 원소(yc, xc)를 선택하는 함수이다. 이때, (ya, xa)와 (yc, xc)는 (y_j, x_j)의 앞과 뒤로 제일 처음으로 NOT_REMOVE의 속성을 가지는 원소이다. 이 세 점을 가지고 $(xa-xc)*(yc-y_j)+(ya-yc)*(x_j-xc)$를 계산한다. 만약 이 값이 0보다 작으면 (y_j, x_j)는 (ya, xa)와 (yc, xc)의 두 점을 잇는 선분보다 바깥쪽에 위치하므로 볼록점으로 간주하고 속성을 NOT_REMOVE로 설정한다. 만약 0보다 크거나 같으면 (y_j, x_j)는 (ya, xa)와 (yc, xc)의 두 점을 잇는

선분보다 안쪽에 위치하므로 오목점으로 간적하고 속성을 REMOVE로 설정한다. 이 점은 앞으로의 검사 대상에서 제외한다. 이러한 과정을 더 이상 제거할 후보점이 없을 때까지 집합 S_1의 "NOT_REMOVE"를 가진 점들에 대해서 반복한다. 이 단계를 S_2, S_3, S_4에도 적용한다.

그림 3에서 G_1={a, b, c}, G_2={d, e, f, g, h, i, j}, G_3={k, l}, G_4={m, n}의 최종 볼록점 결정은 다음과 같다. 각 Gi에서 첫 번째 점과 맨 마지막 점만 최종 볼록점으로 결정한다. 즉, G_1은 a와 c, G_2는 d와 j, G_3는 k와 l, G_4는 m과 n이다.

그림 4는 그림 3에 두 번째 단계를 모두 수행한 결과이다. 입력 패턴의 볼록 헐을 구성하는 점들은 '&'로 표시하였다. 이 점들을 순서대로 선을 연결하면 패턴 'ㄹ'를 포함하는 가장 작은 다각형인 볼록 헐을 얻을 수 있다.

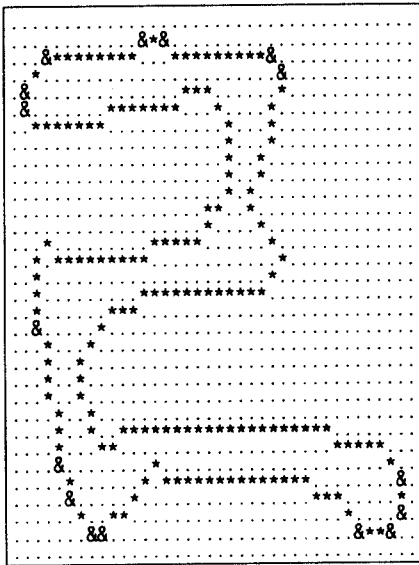



그림 4. 최종적인 볼록 헐

3. 실험 및 결과

이 장에서는 본 논문에서 제안한 알고리즘의 실험 결과 및 성능에 대해서 기술한다. 실험에 사용된 데이터는 PE92 데이터베이스를 사용하였다 [10]. 그림 5는 본 논문에서 제안한 알고리즘에서 추출한 볼록 헐의 몇 가지 예제이다. 볼록 헐을 구성하는 점들은 '&'로 표시하였다.

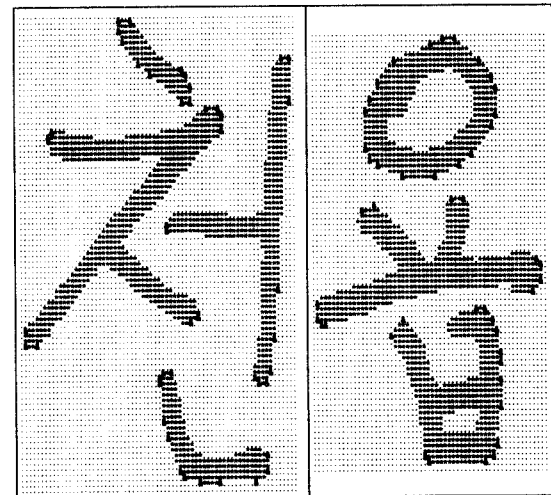
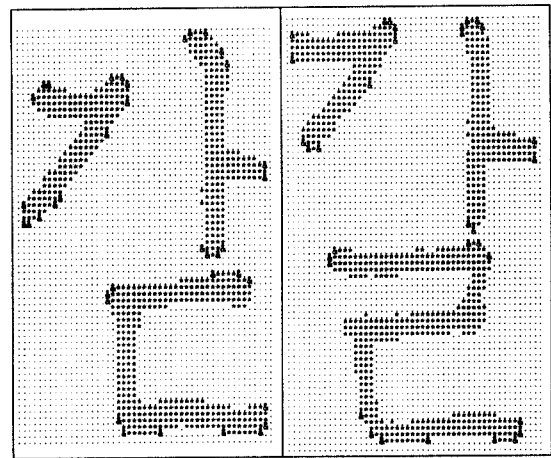
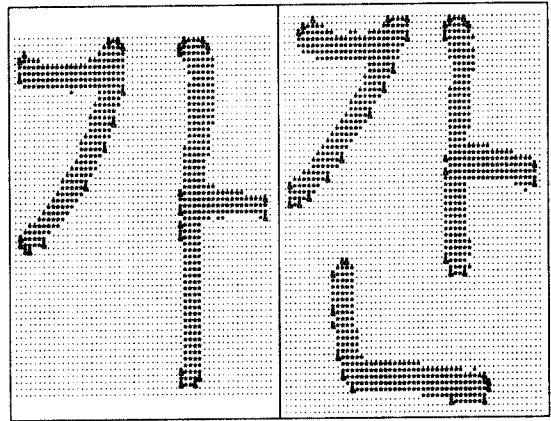



그림 5. 볼록 헐의 몇 가지 예제

또한 본 논문의 알고리즘을 [8,9]의 알고리즘과 수행 속도를 비교하였다. 비교 결과, 표 1에서 보는 바

와 같이 [8]의 방법은 100개의 한글 패턴에 대해서 0.52초, [9]의 방법은 0.71초, 그리고 본 논문의 방법은 0.21초의 수행 속도를 보였다. 또한 실험 결과 분석 중, 데이터가 본래 가지고 있는 특성상 3 가지 방법에서 별로 속도차이가 없는 데이터도 있으며 속도차이가 많이 나는 데이터도 있다. 예를 들어, 그림 5에서 '음'의 초성 'ㅇ'은 대체적으로 외곽선의 모양이 볼록하기 때문에 3 가지 방법에서 속도차이가 거의 나지 않는다. 그러나 '가'의 중성 'ㅏ', 또는 '천'의 중성 'ㅓ'와 같은 패턴들은 외곽선의 모양이 오목한 부분이 많이 포함되어 있다. 이러한 경우, 본 논문의 알고리즘은 첫 번째 단계에서 많은 수의 점들을 제거할 수 있기 때문에 [8, 9] 알고리즘에 비해서 본 논문의 방법이 평균 속도 이상으로 빠르다.

표 1. 알고리즘 수행 속도 비교

알고리즘	수행 속도
[8] 방법	0.52 초 / 100 문자
[9] 방법	0.71 초 / 100 문자
본 논문 방법	0.21 초 / 100 문자

4. 결 론

어떤 물체의 모양을 분석하는 작업은 영상 검색, 사무 자동화, 공장 자동화, 패턴 인식과 같은 응용 분야에 유용하게 사용될 수 있다. 이러한 응용 분야에 적용하기 위해서는 고속의 모양 분석 알고리즘이 필요하다. 그래서 본 논문에서는 모양 분석을 위한 한 가지 방법으로서 2 차원 패턴의 볼록 헐 알고리즘을 제안한다.

본 논문에서 제안한 알고리즘은 크게 후보 볼록점 추출과 최종 볼록점 추출의 두 단계로 나뉜다. 첫 번째 단계에서는 볼록 헐을 구성할 수 있는 후보점들을 추출하는 과정이고, 두 번째 단계는 후보점들 중, 볼록 헐을 구성하지 않는 점들을 모두 제거하는 과정이다.

첫 번째 단계에서는 입력 패턴의 외곽선을 구한 후, 이 외곽선에 패턴을 포함하는 bounding box를 씌워서 겹치는 점들을 그룹으로 나눈다. 그룹과 그룹 사이에 있는 점들의 집합을 세그먼트라고 하면, 외곽선은 여러개의 세그먼트로 나누어진다. 각각의 세그먼트의 양 끝점을 잇는 가상적인 선분보다 안쪽에 위치한 점들을 모두 제거하여 볼록 헐을 구성할 수 있는 후보점을 구할 수 있다.

두 번째 단계는 구해진 후보점들을 대상으로 최종적인 볼록점을 구하는 과정이다. 세그먼트내의 모든 후보점들에 대해서 볼록성을 검사한다. 한 후보점의 볼록성은 양 옆에 위치한 두 개의 후보점들이 이루는 선분을 기준으로 바깥쪽에 위치하는지 아니면 안쪽에 위치하는지에 의해 결정된다. 안쪽에 위치하면 후보점에서 제거한다. 이 과정을 더 이상 제거할 후보점이 없을 때까지 반복한다.

본 논문의 알고리즘을 두 단계로 설정한 이유는 첫 번째 단계에서 볼록 헐을 구성할 수 있는 점들만 남기고 최대한 제거함으로써 두 번째 단계에서 최소한의 연산을 수행하기 위함이다. 이것은 전체적인 알고리즘의 수행 속도를 향상시킬 수 있다. 실험 결과, 본 논문의 방법이 다른 방법보다 속도가 빠름을 알 수 있다.

향후 연구 과제로는 수행 속도 비교에 대한 좀더 객관적인 방법을 선택하는 것이다. 또한 다양한 패턴에 대해서 실험을 하여 패턴이 세 가지 방법에서 어떠한 수행 속도를 가지는 지를 측정할 것이다.

참 고 문 헌

[1] 정규식, 권희용, "내용기반의 인쇄체 영문 문서 영상 검색을 위한 특징기반 단어 검색," *정보과학회 논문지*, Vol.26, No.10, pp.1204-1218. 1999.

[2] W.A. Geisler "Vision system for shape and position recognition of industrial parts," *Developments in Robotics*, North Holland, pp.141-150, 1983.

[3] I.T. Young, J.E. Walker, and J.E. Bowie, "An analysis technique for biological shape," *Inf. & Control*, Vol.25, pp.357-370, 1974.

[4] 김수형, 정선화, 오일석, "필기 한글 문자의 오프라인 인식에 관한 사례 연구," *'98 정보과학회 가을 학술 발표 논문집*, Vol.25, No.2, pp.396-398, 1998.

[5] I.S. Oh, J.S. Lee, and C.Y. Suen, "Analysis of class separation and combination of class-dependent features for handwriting recognition," *IEEE Tr. on PAMI*, Vol.21, No.10, pp.1089-

1094, October 1999.

[6] Y. Zimmer, R. Tepper, and S. Akselrod, "An improved method to compute the convex hull of a shape in a binary image," *Pattern Recognition*, Vol.30, No.3, pp.397-402, 1997.

[7] Jorge Kazuo Yamamoto, "CONVEX_HULL-A pascal program for determining the convex hull for planar sets," *Computer & Geosciences*, Vol.23, No.7, pp.725-738, 1997.

[8] Jack Sklansky, "Finding the convex hull of a simple polygon," *Pattern Recognition Letters*, Vol.1, pp.79-83, 1982.

[9] G.R. Wilson and B.G. Batchelor, "Convex hull of chain-coded blob," *IEE Proceedings of Computers and Digital Techniques*, Vol.136, No.6, pp.530-534, 1989.

[10] 김대환, 방승양, "한글 필기체 영상 데이터베이스 PE92의 소개," *제4회 한글 및 한국어 정보처리 학술발표 논문집*, pp.567-575, 1992.

홍 기 천

1995년 전북대학교 전산통계학과 학사

1997년 전북대학교 전산통계학과 석사

2000년 전북대학교 전산통계학과 박사

2001년~현재 전주교육대학교 컴퓨터교육과 전임강사

관심분야 : 패턴인식, 컴퓨터 비젼 등임.

오 일 석

1984년 서울대학교 컴퓨터공학과 학사

1986년 한국과학기술원 전산학과 석사

1992년 한국과학기술원 전산학과 박사

1992년~현재 전북대학교 컴퓨터과학과 교수

관심분야 : 컴퓨터비젼, 문서 및 문자 인식 등임.